Microsoft Knowledge Base Article - 219905

HOWTO: Dynamically Add and Run a VBA Macro from Visual Basic

Applies To

This article was previously published under Q219905

SUMMARY

When Automating an Office product from Visual Basic, it may be useful to move part of the code into a Microsoft Visual Basic for Applications module that can run inside the process space of the server. This can boost overall execution speed for your application and help alleviate problems if the server only carries out an action when a call is made in-process.

This article demonstrates how to dynamically add a Microsoft Visual Basic for Applications module to a running Office application from Visual Basic, and then call the macro to fill a worksheet in-process.

MORE INFORMATION

The following sample demonstrates inserting a code module into Microsoft Excel, but you can use the same technique for Word and PowerPoint because both incorporate the same Microsoft Visual Basic for Applications engine.

The sample uses a static text file for the code module that is inserted into Excel. You may want to consider moving the code into a resource file that you can compile into your application, and then extract into a temporary file when needed at run time. This would make the project more manageable for re-distribution.

Starting with Office XP, a user must grant access to the VBA object model before any Automation code written to manipulate VBA will work. This is a new security feature with Office XP. For more information, please see the following Knowledge base article:

282830 Programmatic Access to Office XP VBA Project is Denied

Steps to Build the Sample

1. First, create a new text file named KbTest.bas (without the .txt extension). This is the code module that we will insert into Excel at run-time.
2. In the text file, add the following lines of code:

```
Attribute VB_Name = "KbTest"

' Your Microsoft Visual Basic for Applications macro function takes 1
' parameter, the sheet object that you are going to fill.

Public Sub DoKbTest(oSheetToFill As Object)
   Dim i As Integer, j As Integer
   Dim sMsg As String
   For i = 1 To 100
      For j = 1 To 10

         sMsg = "Cell(" & Str(i) & "," & Str(j) & ")"
         oSheetToFill.Cells(i, j).Value = sMsg
      Next j
   Next i
End Sub
```

3. Save the text file to the C:\KbTest.bas directory, then close the file.
4. Start Visual Basic and create a standard project. Form1 is created by default.
5. From the **Project** menu, choose **References**, then select the **Microsoft Excel 10.0 Object Library**, which allows you to use early binding to Excel. For Excel 2000, this is the version 9.0 Type Library, and for Excel 97, this is the 8.0 Library.
6. Add a button to Form1, and place the following code in the handler for the button's **Click** event:

```
Private Sub Command1_Click()
   Dim oXL As Excel.Application
   Dim oBook As Excel.Workbook
   Dim oSheet As Excel.Worksheet
   Dim i As Integer, j As Integer
   Dim sMsg As String

 ' Create a new instance of Excel and make it visible.
   Set oXL = CreateObject("Excel.Application")
   oXL.Visible = True

 ' Add a new workbook and set a reference to Sheet1.
   Set oBook = oXL.Workbooks.Add
   Set oSheet = oBook.Sheets(1)

 ' Demo standard Automation from out-of-process,
 ' this routine simply fills in values of cells.
   sMsg = "Fill the sheet from out-of-process"
   MsgBox sMsg, vbInformation Or vbMsgBoxSetForeground

   For i = 1 To 100
      For j = 1 To 10
         sMsg = "Cell(" & Str(i) & "," & Str(j) & ")"
         oSheet.Cells(i, j).Value = sMsg
      Next j
   Next i

 ' You're done with the first test, now switch sheets
 ' and run the same routine via an inserted Microsoft Visual Basic
 ' for Applications macro.
   MsgBox "Done.", vbMsgBoxSetForeground
   Set oSheet = oBook.Sheets.Add
   oSheet.Activate

   sMsg = "Fill the sheet from in-process"
   MsgBox sMsg, vbInformation Or vbMsgBoxSetForeground

 ' The Import method lets you add modules to VBA at
 ' run time. Change the file path to match the location
 ' of the text file you created in step 3.
   oXL.VBE.ActiveVBProject.VBComponents.Import "C:\KbTest.bas"

 ' Now run the macro, passing oSheet as the first parameter
   oXL.Run "DoKbTest", oSheet

 ' You're done with the second test
   MsgBox "Done.", vbMsgBoxSetForeground

 ' Turn instance of Excel over to end user and release
 ' any outstanding object references.
   oXL.UserControl = True
   Set oSheet = Nothing
   Set oBook = Nothing
   Set oXL = Nothing

End Sub
```

7. For Excel 2002, you must turn on access the VBA project. To do this, start Excel 2002. On the **Tools** menu, point to **Macro**, and then click **Security**. In the **Security** dialog box, click the **Trusted Sources** tab, and then click to select the **Trust access to Visual Basic Project** check box.
8. Run the Visual Basic project.

REFERENCES

For more information on Automation of Office from Visual Basic, please see the Office Development Support site at the following address:

http://support.microsoft.com/support/officedev/

The information in this article applies to:

- Microsoft Office Excel 2003
- Microsoft Excel 2000
- Microsoft Visual Basic Learning Edition for Windows 5.0
- Microsoft Visual Basic Learning Edition for Windows 6.0
- Microsoft Visual Basic Professional Edition for Windows 5.0
- Microsoft Visual Basic Professional Edition for Windows 6.0
- Microsoft Visual Basic Enterprise Edition for Windows 5.0
- Microsoft Visual Basic Enterprise Edition for Windows 6.0
- Microsoft Office XP Developer
- Microsoft Office 2000 Developer
- Microsoft Excel 2002
- Microsoft Excel 97 for Windows
- Microsoft Office Professional Edition 2003
- Microsoft Office Standard Edition 2003

Last Reviewed: 12/15/2003 (4.0)

Keywords: kbAutomation kbhowto KB219905 kbAudDeveloper